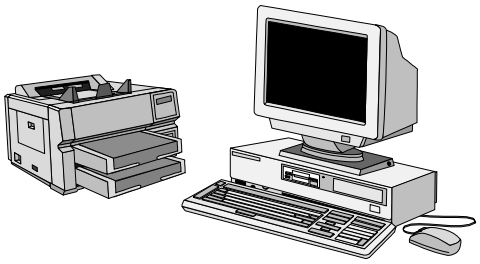
Section 1: Software development process

1.1 Software



This unit is about **software**. What is software? You should already know that any computer system is made up of hardware and software.

The term **hardware** is fairly easy to understand because you can see it. It is all the pieces of equipment that make up the system – the processor, monitor, keyboard, mouse, printer, scanner and so on.

Software is not so obvious. It is all the programs, instructions and data that allow the hardware to do something useful and interesting.

Think about all the different items of software that you have used in the last week or so.

Here is the list of programs that I have used recently:

- **Microsoft Word** (the word processing program that I use – I regularly use three versions of it: Word 2000, Word 98 for MacOS 8, Word v.X for MacOS X)
- **Microsoft Excel** (spreadsheet used to keep charity accounts for which I am the treasurer)
- **ClarisWorks 4** (integrated package – I mainly use its word processor and simple database sections)
- **Internet Explorer** (both PC and Mac versions – for browsing the web)
- **Safari** (web browser for MacOS X)
- three different email clients (**Netscape Communicator**, **MS Outlook** and **Mail**)
- **iPhoto** (for organising my digital photographs)
- **iMovie** (for editing digital movies)
- **Adobe Photoshop** (for editing digital photographs)
- **Citrix ICA** thin client (allows me to connect to my work computer from home)
- **Toast** (for burning CDs)

© Learning and Teaching Scotland 2009

- **Print to pdf** (a shareware program for creating pdf files)
- **Adobe Acrobat** and **Preview** (for viewing pdf files)
- **Macromedia Flash** (for developing animated graphics)
- **Home Page** (an ancient but reliable web page editor)
- some **game** programs
- **Symantec Anti-virus** suite.

But that's not all! On each computer that I have used, a program (or group of programs) called the **operating system** must have been running. So I must add the following to my list:

- **Windows 97** (on the ancient laptop I am using to type these notes)
- **Windows XP** (on another laptop)
- **Windows 2000** (on a computer at school)
- **MacOS 8.1** (on my trusty old Mac clone)
- **MacOS X.2** (on my iMac).

Thirdly, a full list would include all the actual documents, files, web pages, emails and so on that I had accessed, as these are also software. That would be too long a list, so I'll ignore it here.

How about you? Make a list of all the software (programs and operating systems) that you have used over the last few days.

The point about all these programs is this: they didn't grow on trees! They are available for us to use because they have been designed and created by teams of software developers. In this unit, we are going to learn about the process of developing software, and to apply this process to develop some (simple) programs of our own.

1. What is the meaning of the term hardware?

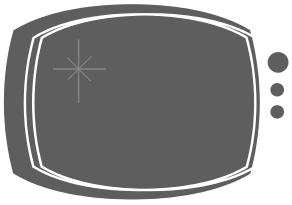
2. Give three examples of software.

© Learning and Teaching Scotland 2009

3. Identify each of the following as either hardware or software:

Item	Hardware	Software
Monitor	✓	
Database		
Windows 97		
Scanner		
An email		
Internet Explorer		
Mouse		
Modem		
Computer game		
Word processor		
Digital camera		

1.2 The development process



Before we think about how software is developed, it is worth considering how any product is developed because the process is essentially the same.

For example, think about the process of developing a new model of television.

Stage 1: Analysis

Before a new product is developed, someone within the company, probably in the marketing department, analyses what people want. They consider which products are selling well, look at what rival companies are producing and maybe even carry out a survey to see what people want. From this they can work out which features are required in their newest model, including its size, target price range and various technical requirements.

They use this information to produce a **specification** for the new model of TV. This states clearly all the features that it must have.

© Learning and Teaching Scotland 2009

Stage 2: Design

The next stage is to turn the specification into a design. Designers will get to work, alone or in groups, to design various aspects of the new TV. What will it look like? How will the controls be laid out? Sketches will be drawn up and checked against the specification. Another team of designers will be planning the internal circuitry, making sure it will allow the TV to do all the things set out in the specification.

Stage 3: Implementation

Once the design phase is over, engineers will get to work to actually build a prototype. Some will build the case according to the design, while others will develop the electronics to go inside. Each part will be tested on its own, then the whole thing will be assembled into a (hopefully) working TV set.

Stage 4: Testing

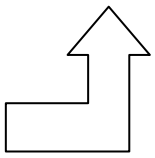
Before the new model can be put on sale, it will be thoroughly tested. A wide range of tests will be carried out.

It might be tested under “normal” conditions. It could be put in a room at normal room temperature, and checked to see that all the controls work correctly, the display is clear, it is nice and stable, and so on.

If it passes this type of testing, it might next be tested under “extreme” conditions. For example, does it still work if the temperature is below freezing, or very hot and humid, if it used for long periods of time, or with the volume or the brightness or contrast set to their maximum values.

Finally, it could be tested under “exceptional” circumstances. What happens if a 2-year-old picks up the remote and presses all the buttons at once? What happens if there is a power cut or a power surge?

If it fails any of these tests, it might be necessary to go back to the implementation (or even design) stage and do some further work, before re-testing.



If it passes all the testing, then it can go into production.

Stage 5: Documentation

However, the development isn’t yet complete! Some documentation will be needed to go with the TV. A user manual

© Learning and Teaching Scotland 2009

containing all the instructions about how to work the new TV, and probably a technical manual for repair engineers.

Stage 6: Evaluation

Once the model is in production, the company will want to evaluate the new product. Does it do what it is supposed to do? Is it easy to use? And, from the engineer's point of view, is it easy to repair?

Stage 7: Maintenance

Stage 6 should be the end of the story, but in the real world there needs to be Stage 7: Maintenance. There are different kinds of maintenance: fixing faults that turn up once it is being used regularly, improving the design to make it even better or making changes for other situations (like making a version that will work in another country).

These seven stages are required for the production process.

Ok, let's see if you have got the idea …

Choose any type of manufactured object – it could be a car, an item of clothing, a ready-made meal, a toy, a piece of furniture, a building, etc.

Now copy and complete this table, writing one sentence to describe each of the seven stages in the production of your chosen object:

Object chosen:

Stage	Description
1. Analysis	
2. Design	
3. Implementation	
4. Testing	
5. Documentation	
6. Evaluation	
7. Maintenance	

© Learning and Teaching Scotland 2009

1.3 A dragon in the dungeon eradicates mice!

Exactly the same process goes into the production of a piece of software. The software engineers and their colleagues carry out all the stages of the software development process in order – analysis, design, implementation, testing, documentation, evaluation, maintenance.

Activity

Consider the production of a new game program by a software company.

Here are descriptions of the seven stages, but they are in the wrong order. Copy and complete another table like the one above, and slot the stages into the correct places:

A. Writing a user guide and technical guide for the software.

B. Deciding what type of game you want to create and what features you want it to have.

C. Adapting the game to run on a different type of computer.

D. Actually writing all the program code.

E. Checking that the program does what it is supposed to do, is easy to use and can be fixed if there is a problem.

F. Working out the details of what the screens will look like, what menus and functions there will be, and other detailed aspects of the program.

G. Getting users to try out the program to make sure it works under most conditions.

Stage	**description**
1. Analysis	
2. Design	
3. Implementation	
4. Testing	
5. Documentation	
6. Evaluation	
7. Maintenance	

© Learning and Teaching Scotland 2009

Check your answers below.

You should have the following:

Stage	Description
1. Analysis	B. Deciding what type of game you want to create and what features you want it to have.
2. Design	F. Working out the details of what the screens will look like, what menus and functions there will be, and other detailed aspects of the program.
3. Implementation	D. Actually writing all the program code.
4. Testing	G. Getting users to try out the program to make sure it works under most conditions.
5. Documentation	A. Writing a user guide and technical guide for the software
6. Evaluation	E. Checking that the program does what it is supposed to do, is easy to use and can be fixed if there is a problem.
7. Maintenance	C. Adapting the game to run on a different type of computer.

© Learning and Teaching Scotland 2009

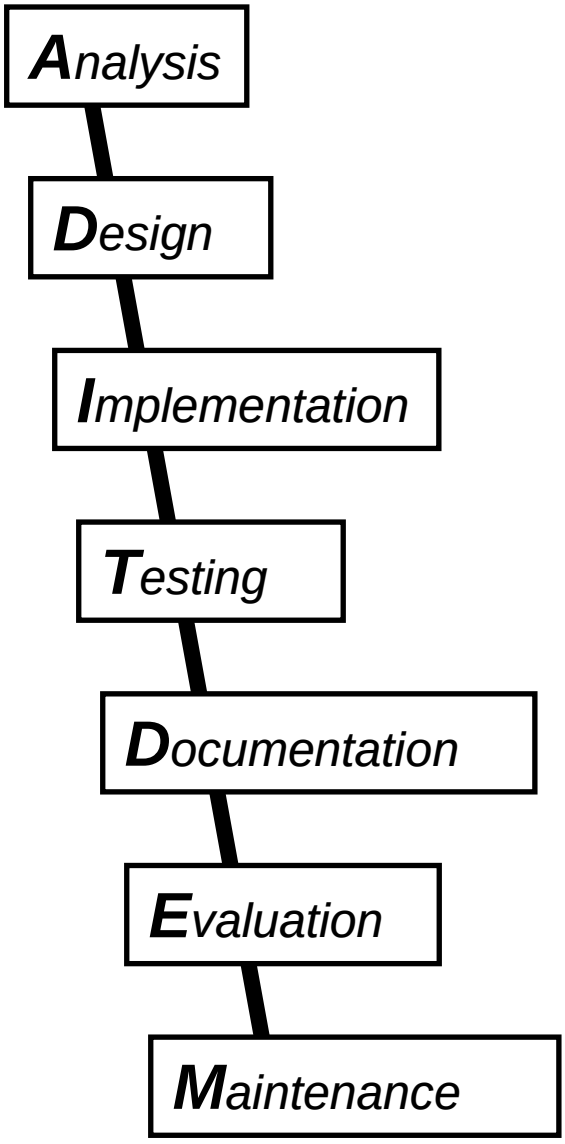
In this course, especially in Section 3 onward, you will be putting this software development process into practice when you produce some simple programs in a high-level computer programming language.

For the moment, it is worth trying to learn the steps in the correct order. I usually use a silly mnemonic for this, like:

A Dragon In The Dungeon Eradicates Mice

which helps me remember ADITDEM:

Analysis
Design
Implementation
Testing
Documentation
Evaluation
Maintenance



You might be able to make up a better mnemonic than this one – so long as it helps **you**, then it's OK!

Next, we will take a closer look at each of the stages.

1.4 Analysis

The main purpose of the analysis stage is to be absolutely clear about what the program is supposed to do. Often, a new program will start from a rough idea. Before getting started, it is important to turn the rough idea into an exact description of how the program will behave. What will it do? What are the inputs and the outputs? What type of computer is it to run on? All these questions, and many more, must be asked and answered at this stage.

© Learning and Teaching Scotland 2009

The result of this is the production of a **program specification**, agreed by both the **customer** (whoever wants the program written) and the **developer** (the person or company who is developing the program).

1.5 Design

Inexperienced programmers are often tempted to jump straight from the program specification to coding, but this is not a good idea. It is worth spending time at the design stage working out some of the important details, including how the program will look on the screen, how the user will interact with the program and how the program might be structured. Program designers use a variety of methods for describing the program structure. Two common ones are called **pseudocode** and **structure diagrams**. There are many others, but we will only consider these two.

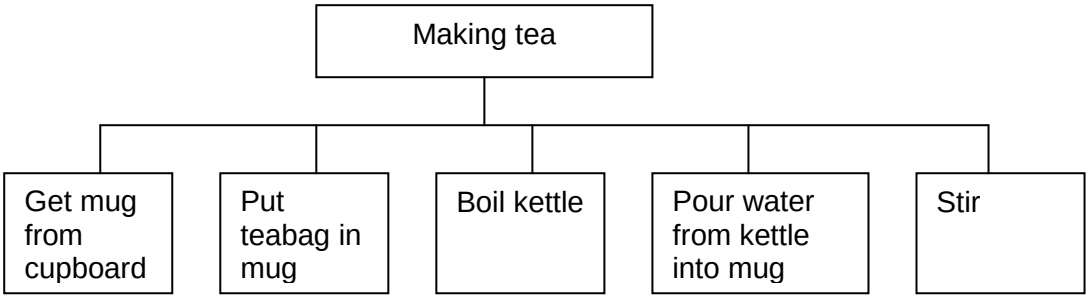
It is easy to understand these if we think about an everyday example, rather than a computer program.

Think about making tea. Here is a list of instructions for this task.

1. Get a mug out of the cupboard.
2. Put a teabag in it.
3. Boil the kettle.
4. Pour boiling water from the kettle into the mug.
5. Stir.

This is an example of **pseudocode**. It is a numbered list of instructions written in normal human language (in this case, English). It doesn't go into all the details, but it gives the main steps.

Another way of showing this is as a **structure diagram**. It could look like this:



Each instruction goes into a separate box. You read **pseudocode** from top to bottom. You read a structure diagram from left to right.

© Learning and Teaching Scotland 2009

Activity

Now try a couple for yourself. Here are some simple tasks.

- Going to school.
- Going to New York.
- Having a shower.
- Phoning a friend.
- Becoming a millionaire.

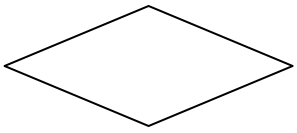
Choose any **two**, and write pseudocode instructions and draw a structure diagram for each one.

Don't make it too complicated. In the tea example, I broke "making tea" into five steps. You could have broken it into many more detailed steps. For example, getting a mug out of the cupboard could be broken down into smaller steps – walk across to the cupboard, open the door, choose a mug, lift it out, close the door, walk back across the room. Try to break the task down into between four and eight steps.

We will use pseudocode in Section 3 when we start to develop our own computer programs.

There are other graphical methods of representing the structure of a program. These include structure charts and flowcharts. Some use a variety of "boxes" to represent different types of instruction. For example, you might see:

to represent a **repeated action**



to represent a **choice**

to represent a step which will be **broken down into smaller steps**

© Learning and Teaching Scotland 2009

1.6 Implementation

In software development, implementation is the process of converting a program design into a suitable programming language.

There are thousands of different programming languages out there, all with their own advantages and disadvantages. For the purposes of this course, you only need to know about two main groups: **machine code** and **high-level languages**. You will learn more about these in Section 2.

1.7 Testing

We thought a bit about testing at the start of this section. Whether we are talking about a new TV, a new item of clothing or a new computer program, the manufacturers will spend a great deal of time in testing. This testing will be carefully planned to test a wide variety of conditions. We can divide this up into three types of testing.

Testing normal conditions

Making sure the program does what it should do when used “normally”.

Testing extreme conditions

Making sure the program can handle situations that are at the edge of what would be considered normal.

Testing exceptional conditions

Making sure the program can handle situations or inputs that it has not been designed to cope with.

You will see examples of all of these in Section 3.

1.8 Documentation

When you buy a product, whether it is a computer program or anything else, you usually get some kind of user guide with it. This tells you how to use the product. It might also contain a tutorial, taking you through the use of the product step by step.

© Learning and Teaching Scotland 2009

Some software comes with a big fat book called user guide or manual; others come with the user guide on a CD.

As well as documentation for the user of the software, there should also be a technical guide of some sort. This gives technical information, which is of little interest to most users, except that it will usually include information about the specification of computer required, including how much RAM it needs, how fast a processor it must have and which operating system is required. The technical guide should also include instructions on how to install the software.

Get hold of a software package that has been bought by your school or college, or one you have bought yourself at home, open it up and take a look inside the box that it came in. Make a list of all the items of documentation that you find there.

1.9 Evaluation

The final stage in the process before the software can be distributed or sold is evaluation. Evaluation involves reviewing the software under various headings to see if it of the quality required.

In this course, we will review software under three headings: fitness for purpose, user interface and readability.

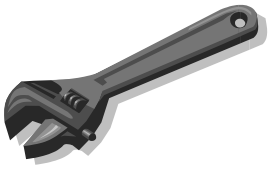
Is the software **fit for purpose**? The answer is “yes” if the software does all the things that it is supposed to do under all reasonable conditions. This means going back to the program specification (produced at the analysis stage) and checking that all the features of the software have been implemented. It also means considering the results of testing, and making sure that the program works correctly and is free from bugs.

The **user interface** should also be evaluated. Is the program easy to use? Is it clear what all the menus, commands and options are supposed to do? Could it be improved in any way?

The third aspect of evaluation that we will consider is **readability**. This is of no direct concern to the **user** of the software, but is important for any **programmer** who may need to understand how the program works. It is to do with the way that the coding has been implemented. Is it possible for the program code to be read and understood by another programmer, perhaps at a later date when the program is being updated in some way? In Section 3 we will see some techniques for improving the readability of a program.

© Learning and Teaching Scotland 2009

1.10 Maintenance



This final phase happens **after** the program has been put into use. There are different types of maintenance that might be required. These are called corrective maintenance, perfective maintenance and adaptive maintenance. You don't need to know these names until Higher level, but it is useful to think about what they mean.

Corrective maintenance means fixing any bugs that appear once the program is in use. Of course, these should all have been discovered during testing. However, most programs (but not the ones you will be writing) are so huge and complex that some bugs are bound to slip through unnoticed. If the bugs are serious in nature, the software company might issue a free "patch" on its website so that users can download the patch and install it with the software, so fixing the bug. If it is a minor bug, they may not bother.

Perfective maintenance is adding new features to the software. These might be suggested as a result of the evaluation stage or they might be suggested by users. These new features will then be added to the software and re-issued as a new version. That's why software often has version numbers.
Each version results from corrective and perfective maintenance of the earlier versions. So, for example, BloggProg 3.2 will be similar to BloggProg 3.1, but with bugs fixed and some new features added.

The third type of maintenance is **adaptive maintenance**. This is where the software has to be changed to take account of new conditions. The most obvious example is when a new operating system comes out. Perhaps BloggProg 3.2 was designed to run under Windows 2000. When Windows XP came along, changes had to be made to BloggProg so that it would work under the new operating system.

© Learning and Teaching Scotland 2009

Questions

1. Match up these descriptions of the stages of the software development process with the correct names (one has been done for you):

Stage	Description
Evaluation	Writing a user guide and technical guide for the software
Testing	Working out the details of what the screens will look like, what menus and functions there will be and other detailed aspects of the program.
Implementation	Deciding what type of game you want to create and what features you want it to have.
Design	Actually writing all the program code.
Documentation	Adapting the game to run on a different type of computer.
Analysis	Checking that the program does what it is supposed to do, is easy to use and can be fixed if there is a problem.
Maintenance	Getting users to try out the program to make sure it works under most conditions.

2. What three criteria will be used for evaluating software in this unit?

3. What is the relationship between pseudocode and a structure diagram?

4. Name two items of documentation usually provided with a software package, and describe what you would expect each one to contain.

5. What three types of testing should be applied to, any software?

6. Describe two examples of maintenance that could be required on a game program.

© Learning and Teaching Scotland 2009